VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Síťový komunikátor v prostředí Microsoft Windows 8 Metro

Network communicator for Microsoft Windows 8 Metro

2014 Richard Juchelka

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Richard Juchelka**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Síťový komunikátor v prostředí Microsoft Windows 8 Metro
Network Communicator for Microsoft Windows 8 Metro

Zásady pro vypracování:

Vytvořte aplikaci pro uživatelské prostředí Microsoft Windows 8 Metro. Aplikace bude využívat aplikační model WinRT, bude spouštěná ve vlastní dynamické dlaždici a bude poskytovat uživateli zpětnou vazbu. Aplikace bude sloužit uživatelům pro komunikaci v rámci firemní sítě. Komunikátor bude splňovat následující požadavky: okamžité odesílaní a doručovaní správ, správa profilu uživatele, správa databáze klientů komunikátoru, možnost cenzury a ochranu před zneužíváním komunikátoru.
Práce bude obsahovat tyto části:
1. Popis vývoje aplikací pro uživatelské rozhraní Microsoft Windows 8 Metro.
2. Analýza a návrh řešení na straně serveru.
3. Analýza a návrh řešení na straně klienta.
4. Implementace a popis implementace řešení komunikátoru.

Seznam doporučené odborné literatury:

[1] Vývoj moderných WinRT-programov pre systém Windows 8 Ján Hanák Microsoft 2012

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Michal Prílepok**

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 20. července 2014 .....

Mé poděkování patří Ing. Michalovi Prílepokovi za trpělivost a ochotu, kterou mi v průběhu práce věnoval.

Abstrakt

Cílem této práce je seznámení s vývojem Windows 8 Metro aplikací a vytvoření vlastního komunikátoru SimpleIM k okamžitému přenosu zpráv pro toto prostředí.

Klíčová slova: Metro, WinRT, Windows 8, Windows Store, bakalářská práce

Abstract

The aim of this work is to introduce the development of Windows 8 Metro applications and create its own communicator SimleIM for immediate transmission of messages for this environment.

Keywords: Metro, WinRT, Windows 8, Windows Store, bachelor thesis

Seznam použitých zkratek a symbolů

WinRT	–	Windows Runtime
C	–	Programovací jazyk C
C++	–	Programovací jazyk C++
C#	–	Programovací jazyk C-sharp
VB	–	Programovací jazyk Visual Basic
IM	–	Instant messanger - okamžité zasílaní zpráv
API	–	Application programming interface - rozhraní pro programování aplikací
IIS	–	Internet information service - internetová informační služba
WNS	–	Windows Push Notification Service
URI	–	Uniform Resource Identifier - jednotný identifikátor zdroje
MVVM	–	Model-View-ViewModel

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Postupem času se trhy zaplavují výkonnými chytrými telefony, tablety a minimalistickými přenosnými počítači. Na to začal Microsoft reagovat a s příchodem Windows 8 ukázal nový směr, jakým se chce tato společnost ubírat. Cílem je postupné sjednocování těchto zařízení, mít možnost jednu aplikaci provozovat jak na klasickém počítači, tak na tabletu či telefonu. Toho chtějí docílit tvorbou aplikací v Metro stylu. Metro je designový styl původně navržen pro aplikace běžící ve Windows Phone, což je operační systém pro chytré telefony. Metro bylo přejmenováno na Windows Store app, ale pro účely práce budu používat dále vývojové označení "Metro". Ve Windows 8 přibylo nové API - Windows Runtime (dále jen WinRT), které umožňuje spouštět aplikace psané v Metro stylu. Microsoft klade na nové API důraz a přizpůsobil pro něj podstatnou část uživatelského rozhraní systému. Nejvíce znatelné to je na předělané nabídce Start, která se stala velkým terčem kritiky uživatelů, jelikož původní Start byl nahrazen oknem, které je výchozí a zobrazí se hned po načtení součástí Windows 8. Tato nabídka zabírá celou plochu obrazovky a obsahuje aktivní dlaždice, které jsou mimo jiné i zástupci pro spouštění aplikací. Tyto dlaždice si představíme v jedné z dalších kapitol. Pro většinu uživatelů je celá myšlenka Microsoftu přivést Metro styl na desktopová zařízení brána negativně. Na druhou stranu je značně patrné, že WinRT aplikace jsou rychlé i na starším hardware. Z pohledu uživatele je ale daleko důležitější práce s takovými programy. Jejich nevýhodou je spouštění v celoobrazovkovém režimu a maximálně dvěma programy spuštěnými vedle sebe. Microsoft ovšem WinRT stále vyvíjí a v dalších verzích bychom se měli dočkat spouštění Metro style aplikací v okně s možností plynulého přepínání mezi okny na ploše.

2 O vývoji Metro aplikací

2.1 Windows Runtime

WinRT je neřízené API pro vytváření aplikací ve stylu Metro. Podporuje vývoj v jazycích C++, C#, VB.NET, JavaScript a TypeScript a umožňuje je propojovat. Podporuje x86 i ARM architekturu a takováto spuštěná aplikace běží v sandboxu, takže má omezené zdroje a přístupy, ale na druhou stranu je tím zajištěna bezpečnost a stabilita. Uživatel musí sám povolit přístup aplikacím, které požadují přístup ke kritickým částem operačního systému a hardware.

Pro Windows Phone 8 existuje vlastní verze WinRT označována jako Windows Phone Runtime. V této práci se zaměřuju na WinRT pro Windows 8.

Windows 8 umožňuje kromě Windows Store app stále spouštět aplikace i pro Win32 nebo CLR (zpětná kompatibilita se staršími systémy). Jak vypadá rozhraní Windows 8 vidíme na obrázku 1(strana 6)[1]

2.2 Uživatelské prostředí

Jak jsem již zmínil, k WinRT lze přistupovat pomocí několika programovacích jazyků. Uživatelské prostředí Windows Store aplikace je možné vytvářet dvěma způsoby - HTML a CSS, nebo pomocí značkovacího jazyka XAML. HTML a CSS se běžně používá pro tvorbu webových stránek. Jsou zde ale omezení na základě použitého jazyka:

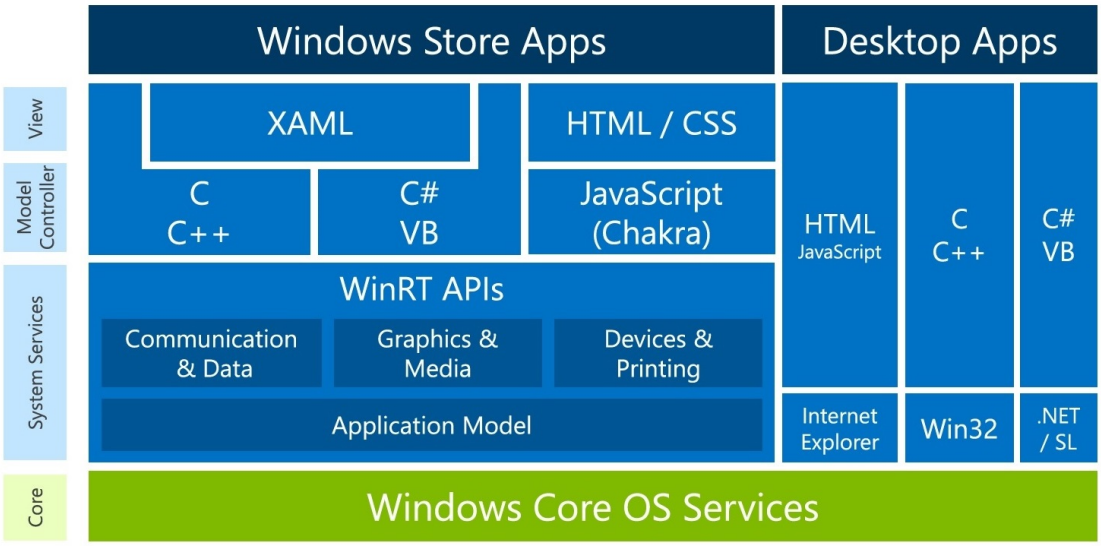
- Pro vytváření view pomocí HTML/CSS je nutné psát logiku v JavaScriptu (TypeScriptu)
- Pro XAML je možné psát logiku aplikace v C/C++/C#/VB

Uživatelské rozhraní je pod WinRT hardwarově akcelerováno. Vývojáři složitých grafických aplikací mohou skrze C++ a XAML nad WinRT využívat i možností DirectX, díky tomu je možné vyvíjet i graficky náročnější hry.

WinRT poskytuje přístup k hardwarovým senzorům zařízení, na kterém aplikace běží. Díky tomu mohou Windows Store aplikace reagovat na spoustu okolností a vstupů od živatele.

Windows 8 je přizpůsobeno pro ovládání na dotykových zařízeních. WinRT aplikace musí být schopna reagovat jak na vstup z běžných zařízení jako klávesnice a myš, tak i na dotyk. K tomu slouží balík nástrojů ve WinRT, které za nás tyto vstupy zajišťuje a vyvolané události zahrnují oba možné způsoby vyvolání. Například událost tlačítka OnClick může být nahrazena událostí PointerPressed, která reaguje jak na kliknutí myší, tak i na dotyk prstem.

Aplikace v Metro stylu jsou na první pohled velmi odlišné oproti aplikacím, na které jsme z desktopových Windows zvyklí. Každá aplikace zabírá ve výchozím stavu celou plochu obrazovky. Klade se důraz na informace a přehlednost, jednoduchost a jednotnost rozhraní. Všechny Metro aplikace si jsou velmi podobné a díky tomu uživatel i v



Obrázek 1: Architektura běhu aplikací ve Windows 8

nové aplikaci hned ví, jak s ní zacházet. Zobrazení je koncetrováno vždy na to, co je aktuálně potřeba uživateli poskytnout. Průchod aplikací připomíná procházení stromem, kdy každá otevřená podsekce se rovná nové stránce. Pro přesun o úroveň zpět se využívá tlačítko se šipkou, které Microsoft doporučuje umístit vlevo nahoru. Příklad Metro stylu můžeme vidět na obrázku 2 (strana 7). Jak je vidět, v daném kontextu aplikace se Microsoft zaměřil na to nejnutnější - okno s konverzací a obrázek uživatele, se kterým se komunikuje. Zároveň si všimněte, že všechny prvky jsou výrazně větší. To je podmínkou Metro aplikací, aby se daly používat i na zařízeních s dotykovým ovládáním.

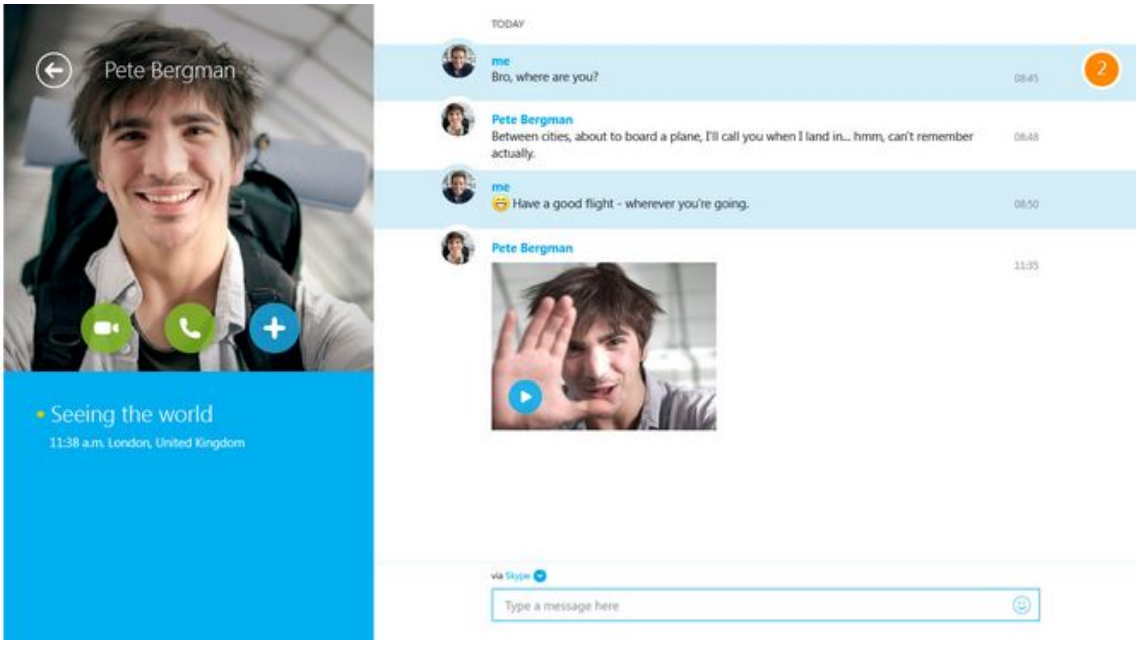
Detailnější informace o uživatelském rozhraní a o interakci s uživatelem popíšu v sekci, kde proberu svou vlastní aplikaci.

2.3 Životní cyklus

Jak jsem již řekl, Windows 8 je připraveno pro použití s dotykovými zařízeními. Takováto zařízení ale nemusí být pouze klasické monitory s dotykovou vrstvou, ale i obrazovky telefonu nebo tabletu. U těchto zařízení je ale problematická výdrž na baterii nebo dostupný výkon.

Ve WinRT je možné mít spuštěno více aplikací zároveň a jako uživatel se nemusíte bát neočekávaného vybití nebo ztráty výkonu zařízení. To díky správě životního cyklu. Systém automaticky pozastavuje a někdy i ukončuje aplikace, které běží na pozadí. Při vývoji je tedy nutné reagovat na tyto události, ukládat stav aplikace, když ji systém pozastavuje a obnovovat stav, když je aplikace znovu spuštěna. Správnou implementací může aplikace vypadat, jakoby byla celou dobu na pozadí spuštěná.

Na obrázku 3 (strana 8)[3] vidíme stavy a přechody mezi nimi.



Obrázek 2: Skype pro Metro od společnosti Microsoft

Aplikace může být pozastavena, jakmile se uživatel přepne na jinou aplikaci, nebo když zařízení detekuje nízký stav baterie. Jakmile je aplikace pozastavena, její stav je udržován v paměti, takže uživatel může rychle a bez odezvy přepínat mezi pozastavenými aplikacemi. V případě, že je potřeba uvolnit operační paměť nebo je kritický stav baterie, systém nejdéle pozastavené aplikace ukončí a vymaže je z paměti.

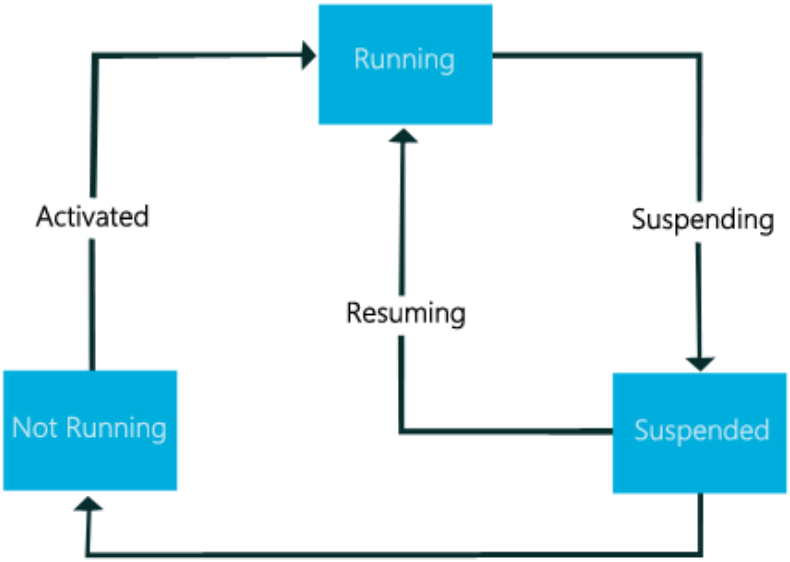
Jakmile uživatel přepne mezi aplikacemi, aplikace v pozadí ještě deset sekund běží, a až poté je pozastavena. Stejně tak i v případě, že uživatel aplikaci ručně ukončí, nebo stiskne klávesovou kombinaci ALT+F4. Během této doby je potřeba zařídit uložení stavu. Aplikace by měla na událost pozastavení vždy reagovat, protože systém aplikaci nedokáže sdělit, kdy bude potřeba ji zastavit.

Daných deset sekund Microsoft zvolil po vyhodnocení statistik, ze kterých vyplynulo, že většina uživatelů se po přepnutí do jiné aplikace během deseti sekund vrátí zpět do aplikace, ze které vyšel. [3]

2.4 Notifikace

Ne vždy je uživatel v naší aplikaci, je ale potřeba ho informovat o změnách nebo o nutnosti nějaké manipulace s naší aplikací, i když je zrovna naše aplikace pozastavena. Pro tento účel jsou k dispozici notifikace. Dva nejčastější typy upozornění jsou Toast a Tile Update.

Toast upozornění je obdelník, který se vysune z pravé horní části obrazovky nad jakoukoliv právě otevřenou aplikací a nese krátkou textovou zprávu. Druhým nejčastějším



Obrázek 3: Životní cyklus aplikací ve WinRT

typem je Tile aktualizace dlaždice, kterou je aplikace reprezentována na obrazovce Start. Na ní se může dynamicky měnit několik upozornění zároveň.

2.5 Přizpůsobení šířce okna

Metro aplikace je navržena tak, aby nezáleželo na velikosti uživatelské obrazovky a aplikace se optimálně přizpůsobila. Na druhou stranu je možné pro každou WinRT aplikaci omezit šířku okna, kterou bude zabírat. To může uživatel řídit chycením aplikace za horní okraj a posunutím na stranu. Takto se aplikace zobrazí přes půl obrazovky. Uživatel dále může toto rozdělení řídit posouváním svislé lišty, která dělí aplikace od sebe. Tato funkčnost už je na programátorovi. Implementace je podobná responzivnímu designu u webových aplikací.

2.6 Distribuce Metro aplikací

Aplikace psané pro Metro se nedají šířit jako samostatné spouštěcí soubory. Microsoft k distribuci vytvořil Elektronický obchod Windows. Pomocí něj uživatelé mohou prohlížet a vyhledávat v katalogu aplikací, rozdělených podle kategorií. Každé aplikaci musí vývojář specifikovat její požadavky a omezení. Například že bude dostupná jen pro určité země a jeho povinností tímto je zajistit, aby byla aplikace pro tyto zákazníky lokalizo-

vána. Pokud aplikace vyžaduje přístup k některým součástem zařízení, musí to být rovněž definováno. Typicky je nutné upozornit, že aplikace vyžaduje přístup k internetu. Uživatel při instalaci programu z elektronického obchodu bude s tímto obeznámen a musí s těmito nároky souhlasit, nebo instalaci zrušit.

Elektronický obchod neboli Windows Store umožňuje prodej aplikací. Pokud budeme chtít uživatelům nabídnout zkušební verzi programu, zajistí to Store sám. Zařídí, aby po uplynulé době byla aplikace zablokována a uživatel ji mohl buď odinstalovat, nebo si zaplatit za její další používání.

3 Síťový komunikátor

3.1 Serverová část

3.1.1 Požadavky

- Funkční požadavky
  - Zabezpečení přístupu klienta k serveru
  - Okamžitá reakce na událost a oznámení klientům
  - Okamžité předání zprávy klientům
  - Ukládání zpráv na serveru
- Nefunkční požadavky
  - Dostupnost v rámci sítě
  - Neustálý provoz

3.1.2 Návrh databáze

Jako databáze bude sloužit SQL Server. Navrženou strukturu tabulek můžeme vidět na obrázku 4 na straně 12.

Dané schéma vyšlo z úvahy, že se v budoucnu doimplementuje navíc i možnost konverzací mezi více klienty. S tímto jsem počítal v duchu celého projektu, i když to nebylo řečeným cílem zadání.

Tabulka Conversation slouží pouze jako prostředník přiřazení zpráv a účastníků konverzace k sobě. Atribut Name - pojmenování konverzace není nutný a je zde pouze kvůli možnému budoucímu rozšíření o hromadné konverzace. Podrobný popis je v tabulce 3.

Tabulka Message reprezentuje samotnou zprávu a její podrobnosti. Její atributy jsou popsány v tabulce 1.

U klienta chci poukázat na atribut sUri, který je potřebný u notifikací klienta. Uri adresa je identifikace zařízení, kam se má notifikace zaslat.

UnreadMessage je druhá vazební tabulka. Spojuje klienta a zprávu. Jakmile je v této tabulce nějaký řádek, je to informace o tom, že daný klient má nepřečtenou určitou zprávu. Pro každou novou zprávu se vytvoří v UnreadMessage tolik řádků, kolik příjemců má v dané konverzaci zprávu obdržet, vyjma odesílatele zprávy. Jakmile systém identifikuje, že některý z klientů již tuto zprávu viděl, vazba tohoto klienta a dané zprávy se z tabulky odstraní. Tím se zajistí, aby se databáze nezahlcovala redundancí zpráv z pohledu každého účastníka a samotná tabulka UnreadMessage se průběžně rychle promazává.

Dále jsou v databázové struktuře dvě samostatné tabulky bez vazeb 5. Jsou jimi tabulka Setting, která obsahuje pouze identifikátor, jméno a bitovou hodnotu. V této tabulce udržuju stavy nastavení.

Druhou tabulkou je BadWord, která udržuje výčet zakázaných slov v textu a tato slova jsou v případě zapnuté cenzury z textu odstraněna.

Název	Datový typ	Velikost	Klíč	Null	Index	Popis
iMessageId	int		PK	ne	ano	ID zprávy
dSentDate	DateTime		ne	ne	ano	Datum odeslání
sText	varchar	max	ne	ne	ne	Text zprávy
dInserted	DateTime		ne	ne	ne	Datum vložení
dLastUpdate	DateTime		ne	ne	ne	Datum změny
iConversationId	int		FK	ne	ano	ID konverzace
iSenderId	int		FK	ne	ano	ID odesílatele

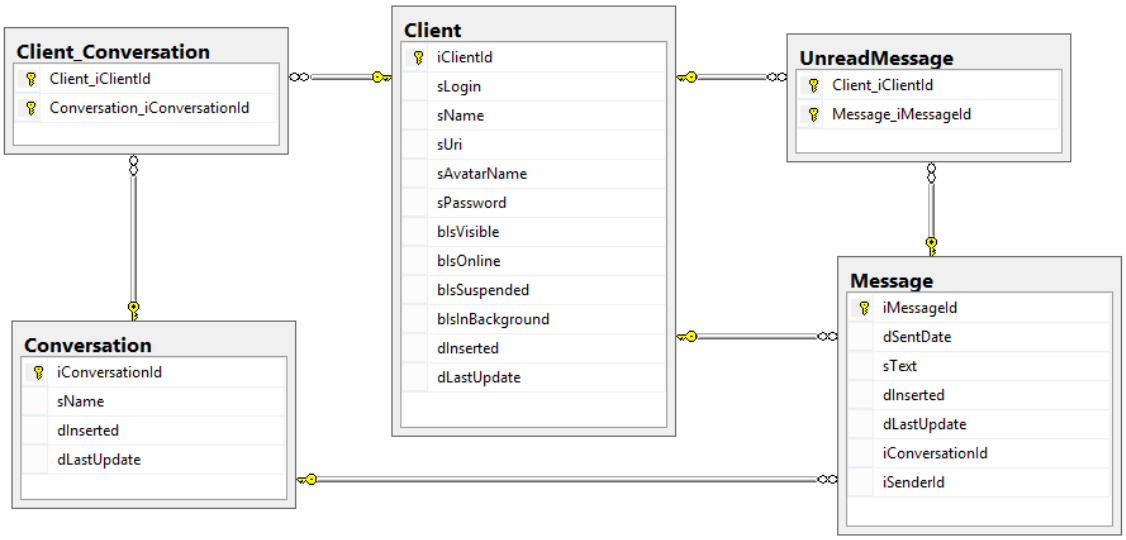
Tabulka 1: Datový slovník tabulky Message

Název	Datový typ	Velikost	Klíč	Null	Index	Popis
iClientId	int		PK	ne	ano	ID klienta
sLogin	varchar	50	ne	ne	ano	Login klienta
sName	varchar	50	ne	ne	ne	Jméno klienta
sUri	varchar	1024	ne	ano	ne	Uri klienta
sAvatarName	varchar	100	ne	ano	ne	Jméno obrázku
sPassword	varchar	64	ne	ne	ne	Hash hesla klienta
bIsVisible	bit		ne	ne	ano	Viditelnost klienta
bIsOnline	bit		ne	ne	ne	Stav klienta
bIsSuspended	bit		ne	ne	ne	Klient pozastaven
bIsInBackground	bit		ne	ne	ne	Klient v pozadí
dInserted	DateTime		ne	ne	ne	Datum vložení
dLastUpdate	DateTime		ne	ne	ne	Datum změny

Tabulka 2: Datový slovník tabulky Client

Název	Datový typ	Velikost	Klíč	Null	Index	Popis
iConversationId	int		PK	ne	ano	ID konverzace
sName	varchar	50	ne	ne	ne	Jméno konverzace
dInserted	DateTime		ne	ne	ne	Datum vložení
dLastUpdate	DateTime		ne	ne	ne	Datum změny

Tabulka 3: Datový slovník tabulky Conversation



Obrázek 4: Návrh tabulek v databázi

Obrázek 5: Dodatečné tabulky bez vazeb

3.2 Implementace služby

Rozhodl jsem se využít Windows Communication Foundation jako API k tvorbě servisní vrstvy, ke které se budou klienti připojovat a za níž bude implementována logika komunikace. Průběh komunikace klientů a služby pro klienty je nastíněn na obrázku 6. Samotná služba běží nad IIS na hostujícím serverovém počítači. WCF službu jsem konfiguroval následovně:

- Instance context mode = PerSession
  - Instancování služby jsem nastavil na PerSession. Což v praxi znamená, že pro každého klienta, který se připojí, se vytvoří nová instance služby a udržuje se po dobu připojení klienta. Jakmile se připojení ukončí nebo přeruší, instance služby zaniká.

  - Tím, že se udržuje spojení, mohu říct, že klient je online (dostupný pro doručení zpráv). A zároveň díky tomu mohu klientovi pomocí callback zavolat metodu ze serveru u klienta a tím mu zaslat zprávu.

- Concurrency = Multiple
  - Tímto jsem nastavil, že pokud klient bude volat na službu více metod zároveň, WCF služba tyto požadavky bude zpracovávat paralelně.
  - Toto jsem uznal jako vhodné pro případ stahování velkého množství obrázků, jelikož v mém návrhu klientské aplikace počítám s dynamickým vykreslováním obrázků jednotlivých kontaktů a pro každý obrázek se vykonává zvláštní dotaz na službu.

WinRT podporuje jen určitý výčet bindingů, se kterými může komunikovat. Jsou jimi:

- BasicHttpBinding
- NetTcpBinding
- NetHttpBinding

Z těchto možností jsem vybral NetHttpBinding, což znamená, že komunikace bude probíhat pomocí Web Sockets. K tomu jsem se uchýlil z důvodů potřeby duplexního spojení, kterým by bylo realné vytvářet spojení i mimo firemní síť, a to BasicHttpBinding neumožňuje. Moje původní myšlenka byla využít NetTcpBinding, ale komunikace přímo přes TCP by mohla být problémová kvůli firewallu klienta a povoleným portům.

Pro účely administrace jsem vytvořil zvláštní službu. Její konfigurace je:

- Instance context mode = PerSession
- Concurrency mode = Single
- Binding = wsHttpBinding

Concurrency mód Single znamená, že pro každého připojeného klienta je vyřazeno jedno vlákno a dotazy se vykonávají postupně.

3.2.1 Objektově-relační mapování

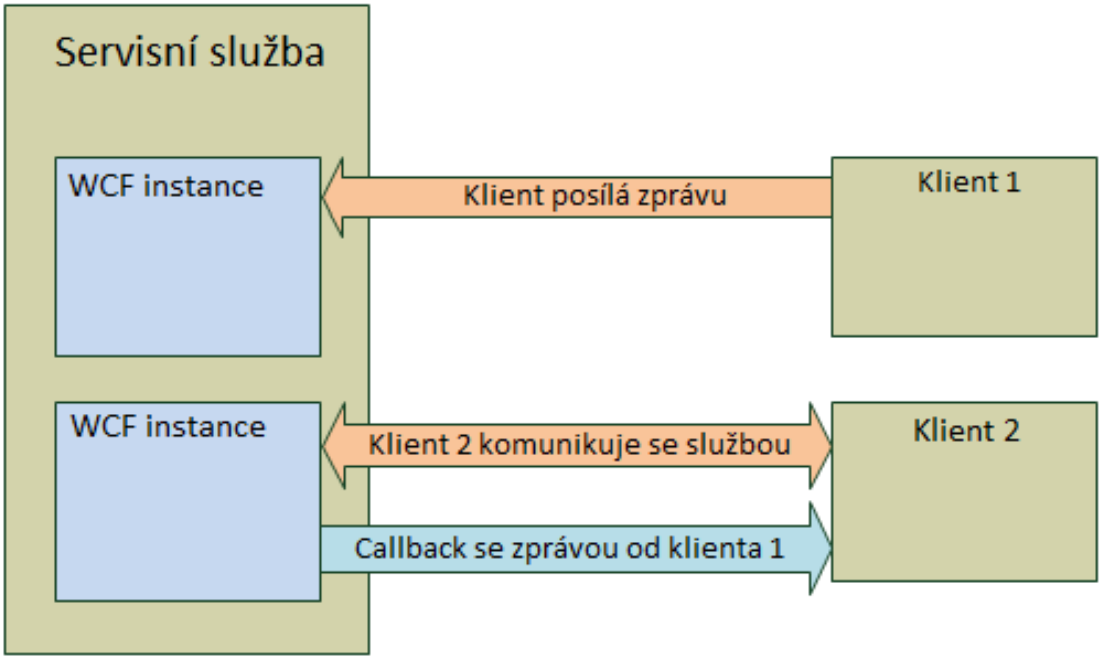
Vedle projektu se službou existuje knihovna HostData_MSSQL. Tato obsahuje objektově-relační mapování, které zastupuje Entity Framework ve verzi 6. Pomocí něj nepřistupuju přímo k databázovým tabulkám, ale využívám jej pouze jako mapování na uložené procedury. Na výpisu kodu 1 je vidět, jakým způsobem jsem tvořil uložené procedury. Všechny parametry všech uložených procedur mají nastaveny výchozí hodnoty. Přímo odkazovaná procedura slouží k načítání stránkovaných zpráv.

Entity Framework mi vytvoří komplexní typ, kterým obalím vrácená data z databáze a ten potom převádím na celkové objekty, se kterými už pracuju v aplikaci. Entity Framework mi nejvíce pomáhá v případě, že mi uložená procedura vrací více výsledků (result setů) najednou (vícekrát se provede příkaz SELECT). Pomocí něj z jednoho výsledku vráceného procedurou mohu načíst postupně další result sety.

Ve stejné knihovně se nachází i namespace HostData_MSSQL.Repository. V tomto prostoru jsou repozitáře, přes které aplikace přistupuje k datům z ORM. V současném stavu jsou to jednuše třídy, které obalují objektově-relační mapování nad databází. Aplikace volá metody těchto tříd za účelem získání nebo uložení již kompletovaných objektů, jejich subkolekcí nebo subobjektů. Pro každý typ je jeden repozitář.

Další knihovnou je HostData_Model. V jejím namespace HostData_Model.Model jsou třídy, jejichž objekty nesou data a reprezentují entity napříč systémem. Jejich třídní diagram je k vidění na obrázku 7. BaseModel je bázovou třídou pro ostatní modely. Obsahuje property UniqueId, která drží ID objektu v databázi.

V této knihovně se také nachází v namespace HostData_Model.Extensions. Obsahuje rozšiřující metody pro práci s IEnumerable objekty.



Obrázek 6: Komunikace klient-server

```
ALTER PROCEDURE [dbo].[GetMessagesFromConversationPaged]
    @p_iConversationId INT = NULL,
    @p_dateFrom DATETIME = NULL,
    @p_messagesCount BIGINT = NULL
AS
BEGIN
-- hodnota 9223372036854775807 je maximalni moznou hodnotou
-- datoveho typu BIGINT a nahrazuje zde symbol '*'
SELECT *
    FROM (
        SELECT TOP (ISNULL(@p_messagesCount, 9223372036854775807))
            M.iMessageId,
            M.iConversationId,
            M.dSentDate,
            M.sText,
            M.iSenderId,
            C.sLogin,
            C.sName,
            C.sUri,
            C.sAvatarName,
            C.bIsVisible ,
            C.bIsOnline,
            C.bIsInBackground AS bIsInBackground,
            C.bIsSuspended AS bIsSuspended
        FROM Message M
        JOIN Client C ON M.iSenderId = C.iClientId
        WHERE
            iConversationId = @p_iConversationId AND
            dSentDate < CONVERT(datetime, @p_dateFrom, 121)
        ORDER BY dSentDate DESC) Message ORDER BY dSentDate;
END;
```

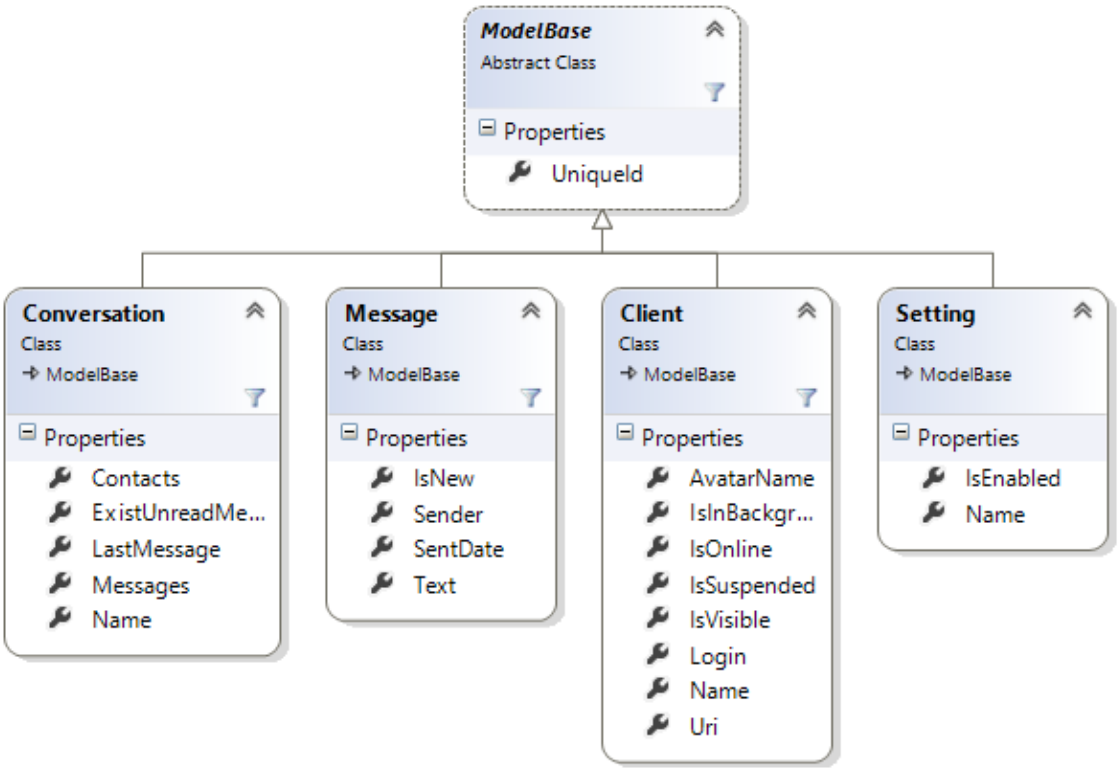
Výpis 1: Uložená procedura pro stránkované načtení zpráv

3.2.2 Vnitřní logika

V této sekci chci zobrazit a popsat zajímavé části logiky implementované za servisní vrstvou.

3.2.2.1 Bezpečnost Pro zabezpečení přihlášení jsem použil asymetrický algoritmus RSA a pro heslo hashovací algoritmus SHA256. Kroky pro verifikaci vypadají následovně:

- Klient se spojí se serverem a vyžádá si jeho veřejný RSA klíč.
- Klient serializuje do Json formátu login a heslo a encryptuje jej pomocí veřejného klíče serveru. Spolu s tím zašle svůj veřejný klíč.
- Server pomocí svého privátního klíče dekóduje zprávu a deserializuje Json. Porovná login a hash hesla s daty v databázi. Pokud souhlasí, uloží si klientův ve-



Obrázek 7: Třídní diagram tříd reprezentujících entity

řejný klíč a vygeneruje token. Token se generuje spojením loginu klienta a aktuálního času a následně se textový řetězec zahashuje algoritmem SHA256. Token se enkóduje klientským veřejným RSA klíčem a vrátí klientovi.

- Klient si token dekóduje pomocí svého privátního klíče a encryptuje jej serverovým veřejným klíčem. Takto si klient předpřipraví token na další komunikaci.

S každým voláním servisní metody klient musí zaslat token. Pokud jej nezašle nebo není správný, server bude volání ignorovat a nebude klientovi vracet žádná data.

3.2.2.2 Notifikace Notifikace, které negeneruje samotná WinRT aplikace, ale jsou vytvářeny externě z cloudu, se označují jako Push Notifikace. Microsoft k tomuto účelu provozuje službu na svých serverech, která zajišťuje a zprostředkovává zasílání notifikací do klientského zařízení. Nazývá se Windows Push Notification Service (dále jen WNS).

Pro využívání této služby je ale potřeba mít svou vyvíjenou aplikaci registrovanou na Windows Store Dashboard, což je místo pro vývojáře, kde spravují a publikují svou aplikaci. Po registraci aplikace získáme Package Security Identifier (dále jen SID) a tajný klíč. Každá aplikace má své vlastní jedinečné přihlašovací údaje a není možné s těmito údaji zasílat notifikace do jiné WinRT aplikace.

Následuje postup, jakým se musí projít, aby byly notifikace zprovozněny. Grafické znázornění je na obrázku 8.

- WinRT aplikace vyšle požadavek k získání push notifikačního kanálu na Notification Client Platform.
- Notification Client Platform vytvoří požadavek na WNS k vytvoření notifikačního kanálu. Tento kanál je vrácen do zařízení ve formě URI.
- Notifikační kanál URI je vrácen do WinRT aplikace.
- WinRT aplikace zašle servisní vrstvě získané URI. V tuto chvíli je zařízeno vše pro zasílání notifikací.
- Naše služba se pomocí SID a tajného klíče autentifikuje na serverech Microsoftu odesláním SID a tajného klíče jako POST požadavek přes zabezpečený kanál SSL. Odpovědí je token, který se bude odesílat s každou notifikací a naše služba se jím autorizuje. Poté, když bude chtít odeslat klientovi notifikaci, vykoná to dotazem na WNS pomocí URI.
- WNS příjme požadavek a nasměruje notifikaci do klientského zařízení.

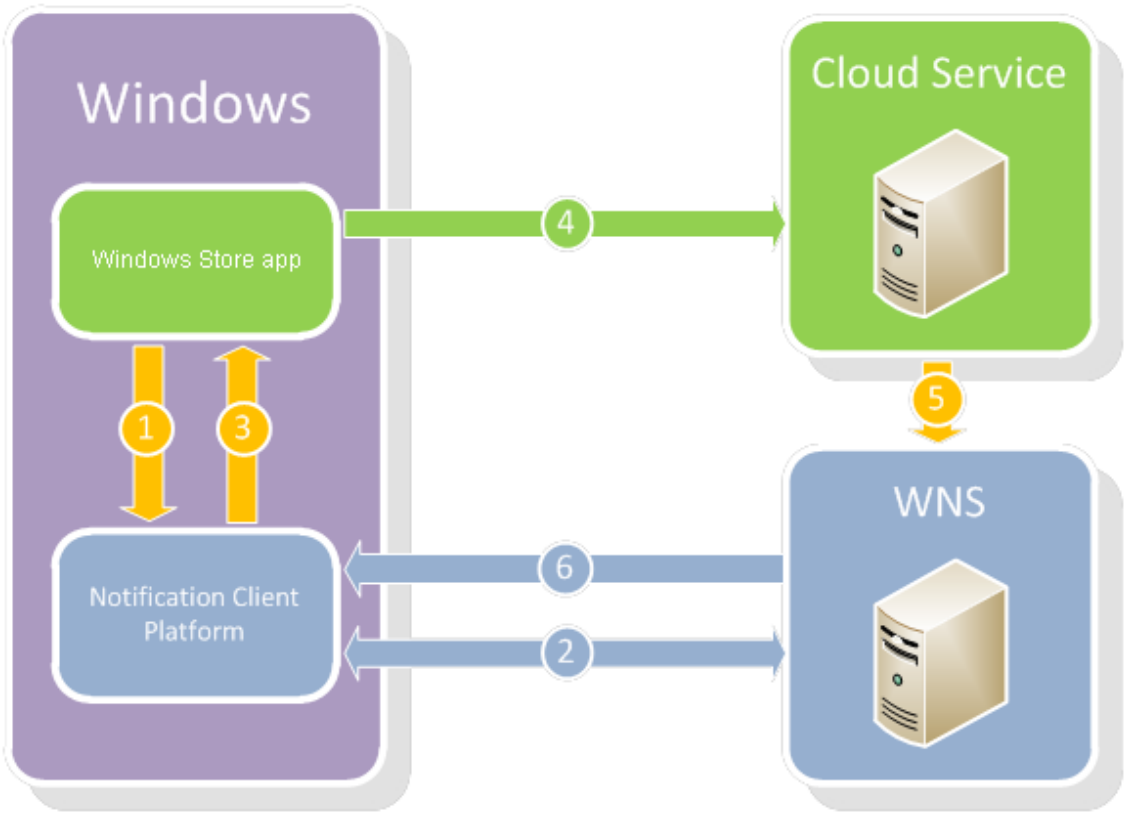
3.2.2.3 Přeposílání zpráv Další důležitou částí je samotná logika zasílání zpráv. Uložil jsem ji do namespace SimpleIM_Server.CommunicationLogic. Ve třídě Communication se udržují připojení klienti, jejich callback kanály a hlavně je zde metoda Whisper(), která se stará o doručení zpráv příjemcům. Jak vypadá proces poslání nové zprávy je vidět v sekvenčním diagramu 9 na straně 19.

Pro upřesnění, každá zpráva, kterou klient A odešle, se odesílateli zobrazí až po návratu ze serveru. Je to kvůli možné cenzuře, aby i odesílatel viděl, v jaké podobě uvidí jeho zprávu příjemce.

3.3 Klientská část

3.3.1 Požadavky

- Funkční požadavky
  - Přihlášení klienta do systému
  - Přijímání zpráv od jiných klientů
  - Odesílání zpráv
  - Vytváření nových konverzací
  - Nastavení vlastního obrázku k účtu
  - Odhlášení klienta ze systému
  - Reakce na systémové události pozastavení a obnovení běhu aplikace



Obrázek 8: Zasílání notifikací přes WNS

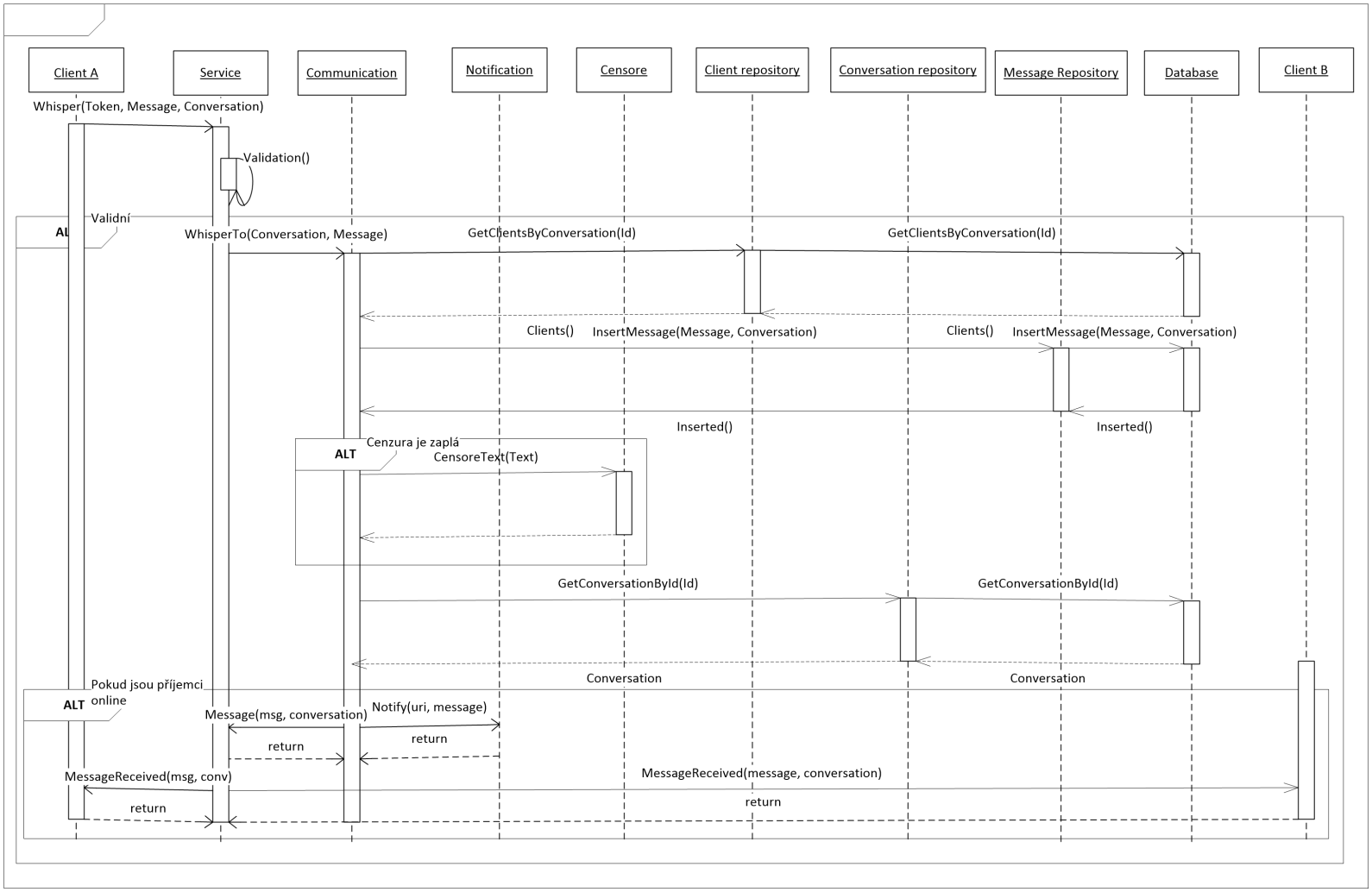
  - Schopnost reakce na změnu šířky obrazovky a přizpůsobení uživatelského rozhraní
  - Schopnost notifikovat uživatele o nastálých událostech

- Nefunkční požadavky
  - Rychlé spuštění
  - Nutnost odezvy a žádné blokování rozhraní delšími procesy v systému
  - Přizůsobení pro běh nad WinRT

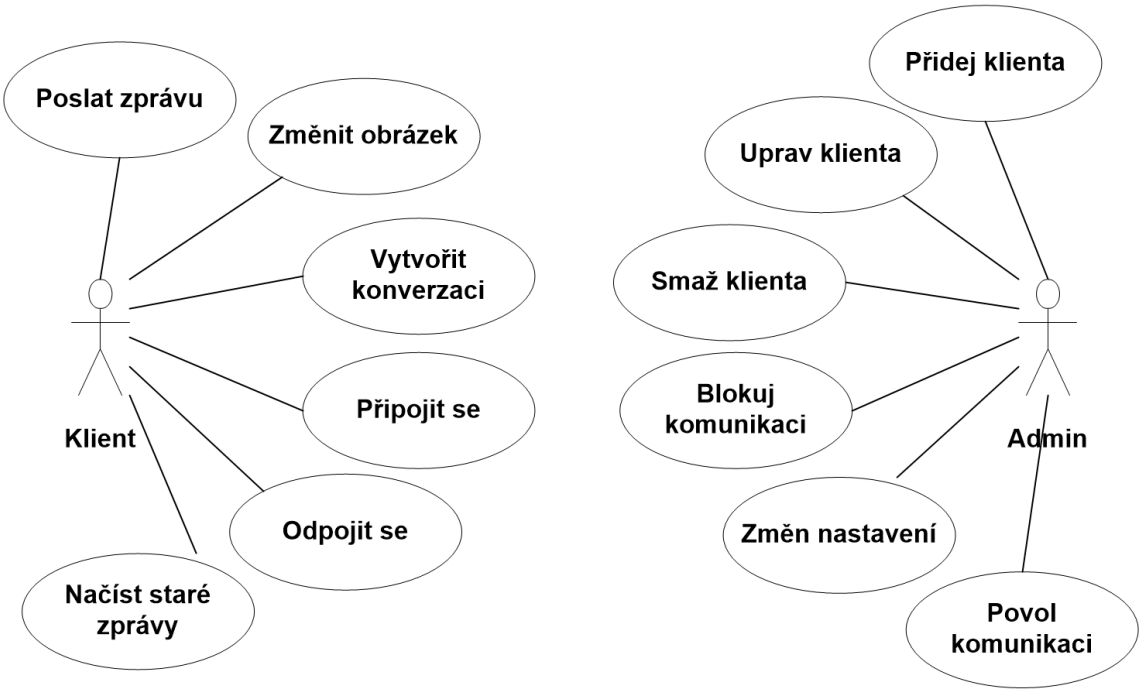
3.3.2 Případy užití

Diagram případů užití je vidět na obrázku 10 na straně 20
Přihlášení uživatele do stavu online

- Název: Přihlášení uživatele
- Aktéři: Uživatel



Obrázek 9: Sekvenční diagram zasílání zprávy



Obrázek 10: Use case diagram

- Prekondice: -
- Hlavní průběh:
  - 1. Uživatel klikne na přepínač stavu
  - 2. Systém se spojí se serverem a proběhne validace.
  - 3. Uživateli se zobrazí konverzace a seznam kontaktů.
- Rozšíření:
  - 3a Uživateli se zobrazí chyba o připojení k serveru.
  - 3b Uživateli se zobrazí chyba o špatných přihlašovacích údajích.
    - 3b1 Systém zobrazí postranní panel s formulářem pro zadání přihlašovacích údajů.

Vytvoření nové konverzace

- Název: Vytvořit konverzaci
- Aktéři: Uživatel
- Prekondice: -

- Hlavní průběh:
  - 1. Uživatel klikne na ikonu zastupující jiného klienta.
  - 2. Systém zkontroluje existující konverzace.
  - 3. Uživateli se zobrazí stránka s detailem nové konverzace.
- Rozšíření:
  - 3a Uživateli se zobrazí stránka s detailem již existující konverzace s požadovaným uživatelem.

Odpojení uživatele ze systému (přepnutí do stavu offline)

- Název: Odpojit se
- Aktéři: Uživatel
- Prekondice: -
- Hlavní průběh:
  - 1. Uživatel klikne na přepínač stavu.
  - 2. Systém zpracuje požadavek odesláním příkazu k odhlášení.
  - 3. Uživateli se zobrazí domovská stránka aplikace již bez kontaktů a konverzací.

Zobrazení zpráv z historie konverzace

- Název: Načíst staré zprávy
- Aktéři: Uživatel
- Prekondice: Uživatel má otevřenou konverzaci.
- Hlavní průběh:
  - 1. Uživatel posune posuvník v okně se zprávami nahoru.
  - 2. Systém si vyžádá starší zprávy
  - 3. Uživateli se dynamicky přidají starší zprávy do okna s konverzací.
- Rozšíření:
  - 3a Uživateli se zobrazí všechny dostupné zprávy bez možnosti posunovat se zpět.

Odeslání nové zprávy do systému

- Název: Odeslat zprávu

- Aktéři: Uživatel
- Prekondice: Uživatel má otevřenou konverzaci
- Hlavní průběh:
  - 1. Uživatel napíše zprávu a potvrzením odešle
  - 2. Systém sestaví seznam připojených příjemců a těm zprávu odešle.
  - 3. Uživateli se zobrazí odeslaná zpráva.

3.3.3 Uživatelské rozhraní

Jak jsem vysvětlil v úvodu, Metro aplikace musí být pro uživatele jednoduchá a má obsahovat minimum prvků dostatečně velkých pro dotyk v kontrastu se zobrazenými informacemi, na které je kladen důraz. Aplikace má dvě stránky a dva layouty nastavení.

Výchozí stránka je pojmenována jako Home.xaml. Po spuštění je prázná, zobrazuje pouze sekce Konverzace a Kontakty. Data se zobrazí až po přihlášení uživatele do systému. To se provede přepnutím spínače v pravém horním rohu obrazovky. Po zapojení do systému se načte historie konverzací a všichni dostupní klienti. Jak to vypadá je vidět na obrázku 11. Pro úsporu místa je na obrázku screen domácí obrazovky zároveň i v režimu zobrazení půlené stránky. Pokud uživatel ručně posunutím svislé lišty ještě více zúží prostor pro aplikaci, odstraní se všechny prvky vyjma historie konverzací (Obrázek 12). Zároveň si všimněte oranžového bodu, který znamená, že v této konverzaci je nová, nepřečtená zpráva.

Druhá stránka obsahuje samotnou konverzaci. Jmenuje se ChatPage.xaml a obsahuje v levém sloupci historii konverzací a ve zbytku prostoru detail vybrané konverzace s účastníky diskuze, historií zpráv a formulářem pro vytvoření nové zprávy (Obrázek 13). Na obrázku 14 je vidět jak se aplikace přizpůsobí maximálnímu zúžení. Zůstane pouze to hlavní - historie zpráv a možnost vytvoření zprávy nové.

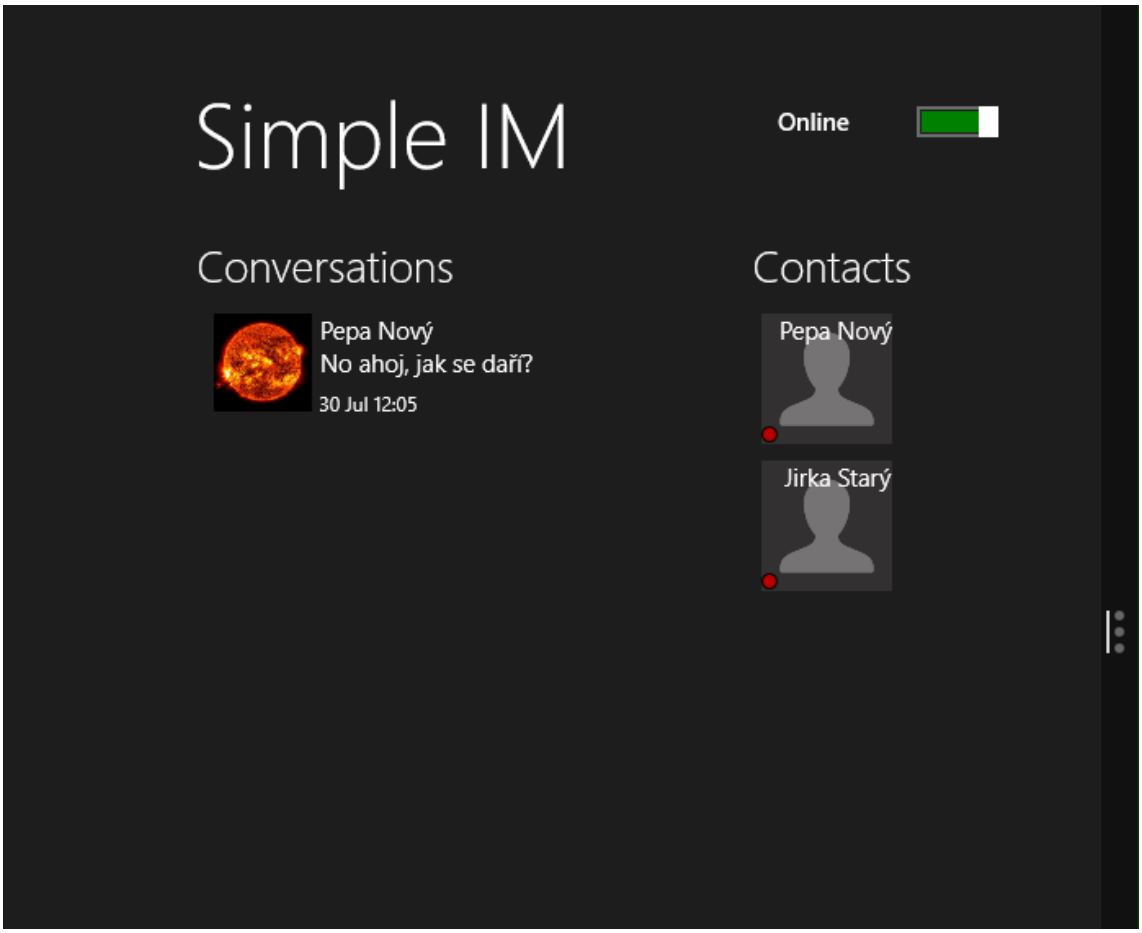
V seznamu kontaktů je u každého klienta kolečko, jehož barevná výplň indikuje jeho stav. Zelená značí jeho dostupnost online, červená je ve stavu offline.

Tile notifikace (notifikace zobrazená uživateli přímo na dynamické spouštěcí obrazovce) se zašle všem příjemcům zprávy, vyjma odesílatele. Na dlaždici se může průběžně střídat více zpráv. Jak prakticky v komunikátoru aktualizace na dlaždici vypadá, je vidět na obrázku 15.

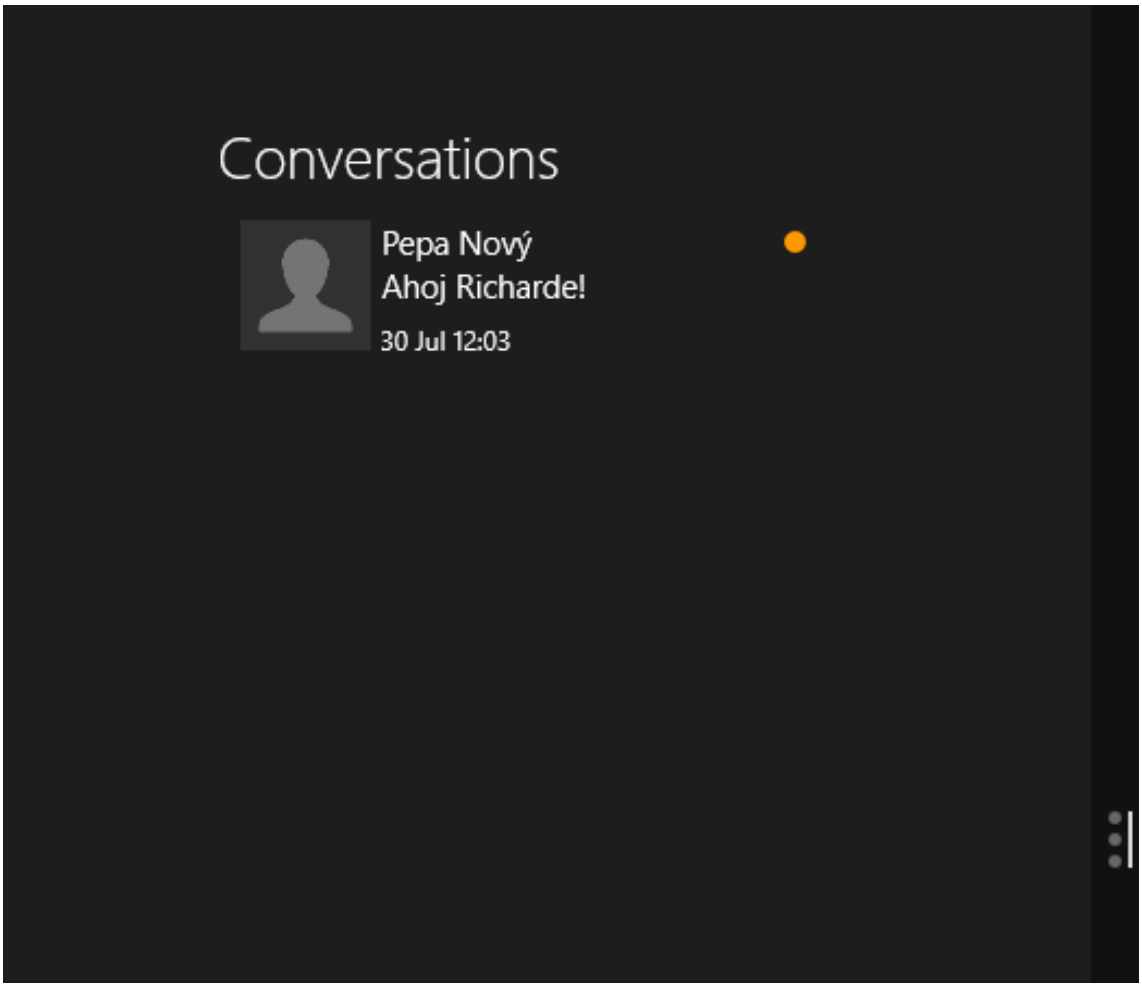
Toast notifikace je druhým typem notifikací, které ve své aplikaci využívám. Server tuto notifikaci zašle všem příjemcům nové zprávy vyjma odesílatele, jejichž aplikace je buď pozastavena, nebo není zrovna zobrazována uživateli. Toast notifikace je vidět na obrázku 16.

3.3.4 Implementace

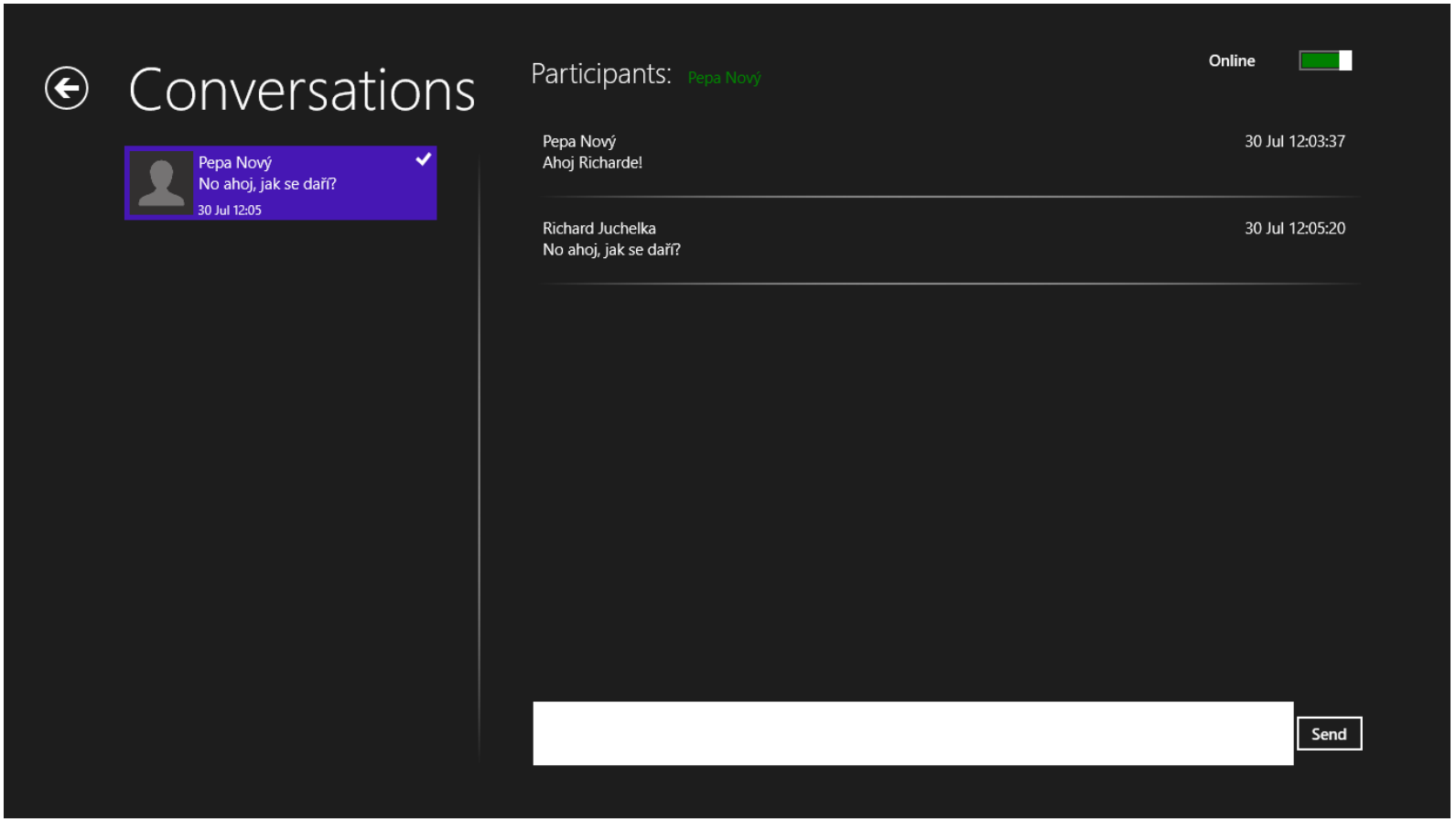
Rozdělení projektu klienta do adresářů:



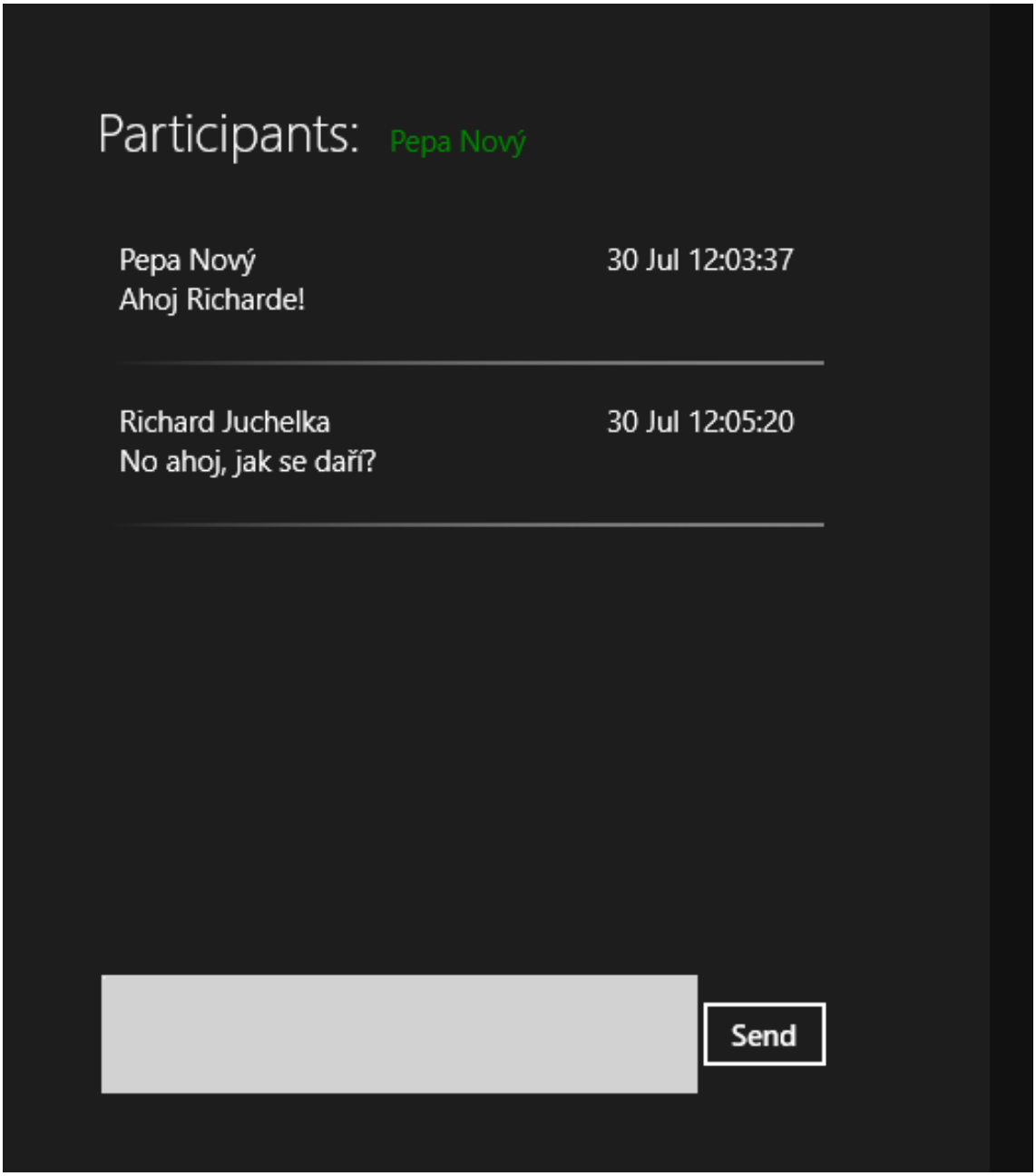
Obrázek 11: Zobrazení výchozí stránky po připojení. Takto se aplikace přizpůsobí zobrazení na půl obrazovky.



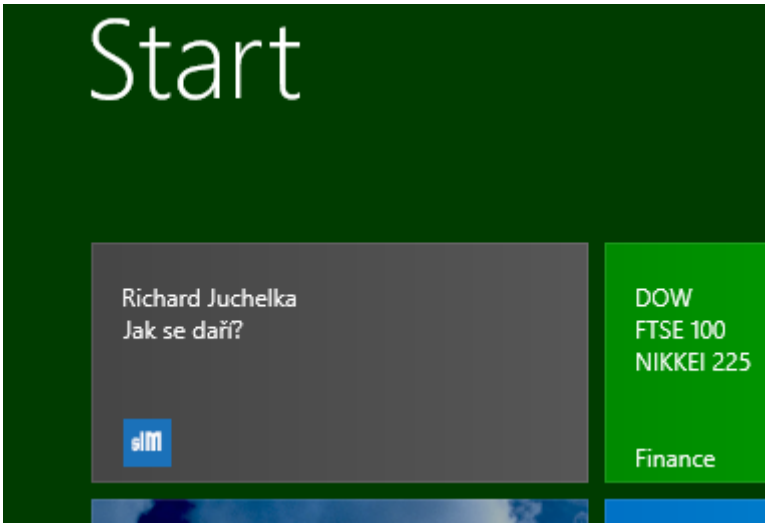
Obrázek 12: Zobrazení výchozí stránky po připojení. Takto se aplikace přizpůsobí zobrazení na třetinu obrazovky.



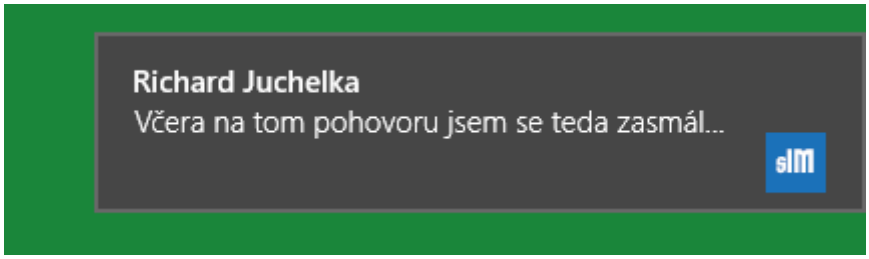
Obrázek 13: Chatovací stránka v plném náhledu



Obrázek 14: Chatovací stránka v maximálním zúžení.



Obrázek 15: Zobrazení notifikace na aktivní dlaždici.



Obrázek 16: Zobrazení toast notifikace.

- Assets
  - Obsahuje obrázky a loga aplikace v různých velikostech. Tato loga Windows aplikuje jako vzhled dlaždice na obrazovce Start, náhled aplikace v pozadí, ikonku u notifikace apod.
- Common
  - Zde jsou třídy, které využívá celý projekt. Typicky mnou implementované observable kolekce IncrementalObservableCollection, SortableObservableCollection a ObservableDictionary. Dále extension metody ve třídě Extensions, konvertory pro view, vlastní ServiceProxy třída, obslužná třída pro pozastavení aplikace a další.
- DataModel
  - Zde jsou modely a viewmodely. ViewModely komunikují se servisní vrstvou skrze ServiceProxy z adresáře Common.
- Security
  - Sem jsem ukládal třídy, ve kterých implementuju metody pro algoritmus RSA a SHA.
- View
  - Zde jsou XAML stránky a jejich codebehindy. V podsložce Settings jsou layouty s view pro nastavení aplikace.

Zajímavou částí, kterou chci popsat, je implementace stránkovaného načítání historie zpráv. Logika je implementována ve třídě IncrementalObservableCollection.

Při jejím vytvoření je nutné v generických typech zadat Model, který bude kolekce udržovat a taky její ViewModel, který se bude starat o načítání ze serveru. O volání načtení dat se starají metody GetLastItemsAsync, která zařídí stažení posledních dat (využívám ji pro stažení posledních zpráv v konverzaci) a GetPreviousItemsAsync, která již data stránkuje. Ta získá v parametru ID konverzace, poslední načtený objekt a počet dalších objektů k získání. Tato metoda se volá v případě, kdy se klient posuvníkem přiblíží v historii zpráv k hornímu okraji. Metoda ihned z daného viewmodelu zavolá metodu na načtení dat a přidá je do kolekce. Nové zprávy se přidají do kolekce a ve view se plynule zobrazí. Kolekce si udržuje i bitový příznak o tom, jestli existují další data v databázi a pokud ne, volání o stažení dat se zablokuje.

Každý klient si v layoutu nastavení, do kterého se dostane, pokud myší sjede do pravého dolního rohu a vybere symbol nastavení, může nahrát svůj obrázek. Ten se zmenší na rozměr 200 x 200 pixelů a nahraje na server, kde se uloží do adresáře App_Data/Images.

Klient pomocí partial class přidává do třídy Client, jíž definice je za servisní vrstvou, property typu BitmapImage. Při volání get části se v případě, že je property prázdná,

stáhne obrázek ze serveru. Tím jsem zajistil dynamické načítání jednotlivých obrázků. Vykreslují se postupně, jak se každý zvlášť načte.

4 Administrace

Administraci jsem psal jako poslední, a díky tomu jsem zde zúročil nabyté zkušenosti z tvorby v modelu MVVM. View je psané v technologii WPF.

4.1 Uživatelské rozhraní

Administrace má dvě okna. Přihlášení do systému, a poté hlavní okno s úlohami nad systémem. Na obrázku 17 je vidět přihlašování. Vstup pro heslo je omezeno na minimálně čtyři znaky, až poté je možné stisknout tlačítko pro přihlášení. Stav přihlášení se zobrazuje v textu pod tlačítkem. Pokud dojde k chybě, zde se chyba, s důvodem proč k ní došlo, červeně vypíše. Po úspěšném přihlášení se zobrazí okno s funkcemi pro administrátora (obrázek 18).

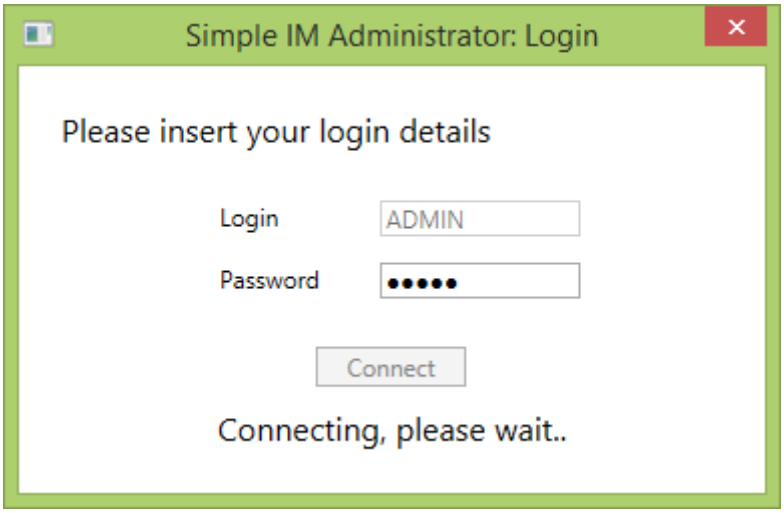
V levé části nahoře je seznam všech klientů. Obsahuje náhledové informace o jménu, loginu, stavu jeho přihlášení a zda je pro ostatní klienty viditelný. Pokud označíme některého ze seznamu, je možné mu v pravém sloupci změnit jméno, heslo nebo viditelnost. Pod editačním polem je pole s možností přidání nového uživatele. Zde je opět možné zadat jméno, heslo a jeho viditelnost. Login systém generuje sám a stará se o jedinečnost loginu. Login se skládá z prvních tří znaků jména a dále tří znaků čísla. Číslo se inkrementuje vždy o jedno, pokud již existuje login začínající stejnými třemi znaky. V oblasti s nastavením vlevo dole se zobrazují zaškrtávací políčka pro aktivaci nebo deaktivaci funkce. Nutno podotknout, že v případě, že se do systému přidá nová funkce, není potřeba upravovat administrátorskou aplikaci, protože políčka se generují dynamicky podle příchozích dat.

V pravé spodní části je prostor na další funkce, aktuálně je zde jen tlačítko pro refresh dat, které administrátor zobrazuje.

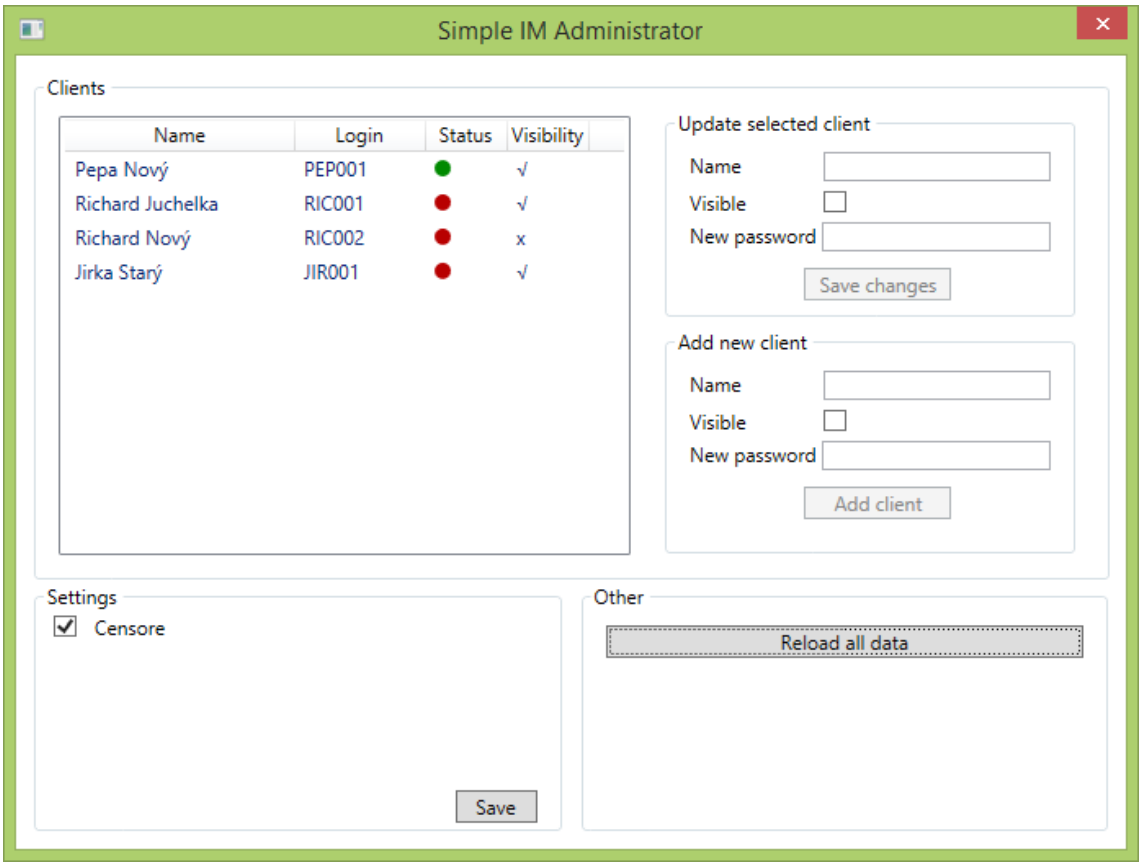
4.2 Implementace

Celá administrátorská aplikace je psaná v modelu MVVM. To znamená, že ve view se prvky bindují na property a kolekce, které jsou ve ViewModelu a v codebehindu jsou jen nejnutnější věci jako nastavení data kontextu pro view (namapování instance viewmodelu na kontext stránky), a v mém případě ještě implementace funkcí, které se mají stát, pokud je Action z viewmodelu aktivována (např. přechod na další stránku).

Namísto událostí, které aktivní prvky ve view mohou spouštět, jsem využíval Command. Commandy v mém řešení jsou vstupní mechanismy, které vytvoří cestu mezi např. stisknutým tlačítkem a metodou, která se vykoná ve ViewModelu. Command má navíc schopnost ovliňovat prvky tím, že se mu nastaví jedna metoda, která vrací boolean v závisloti na tom, jestli jsou splněny podmínky pro to, aby se mohl samotný command spustit. Pokud ne, nebude např. na zmíněné tlačítko možné kliknout. Jakmile ale bude podmínka splněna, po kliknutí se vykoná již samotná úloha. Tímto odpadají implementace událostí v codebehindu view a kód je čitelnější.



Obrázek 17: Přihlašovací okno do administrace.



Obrázek 18: Okno s funkcemi pro administrátora.

MVVM se dá implementovat více způsoby. Já použil verzi, kdy je na jednu stránku view jeden viewmodel. Tento viewmodel žádá o data přes třídu ServiceProxy, která je opět v adresáři Common.

V adresáři common se nachází dále:

- Konvertory (soubor Convertors.cs). Obsahuje implementace tříd, které dědí z rozhraní IValueConverter a metody těchto tříd se používají ve view pro převedení hodnot z bindovaných vlastností na hodnoty použitelné v XAMLu.
- EventArguments. Zde jsou třídy používané jako typy argumentů.
- Extensions. Obsahuje rozšiřující metody.
- RangeObservableCollection je implementací IObservableCollection, rozšířené o možnost přidávání více objektů najednou do kolekce s pouze jedním spuštěním události změny kolekce.
- RelayCommand implementuje ICommand a jeho instance využívám pro routování funkcí XAML prvků do ViewModelu.
- SecuritySHA a SecurityRSA je implementací těchto algoritmů pro mé účely.
- ServiceProxy obaluje komunikaci mezi servisní vrstvou a ViewModelem. Vytváří a ukončuje spojení a reaguje na zavření nebo pád spojení.

5 Zprovoznění aplikace

- Serverová část se musí nahrát na IIS.
- Clientské aplikaci a administrátorské aplikaci se musí nastavit service reference na server.
- V administrátorské aplikaci je potřeba se ujistit, zda existují nějací klienti a jsou viditelní. Případně je přidat. Heslo do admina je "Admin".
- Pro otestování klienta je nutné mít počítač s Windows 8.1 Update 1 a spouštět jej ve Visual Studio 2013 Update 2, jelikož jsem aplikaci nechtěl publikovat ve Windows Store. Pokud na testovaném zařízení nebudou tyto verze Windows a Visual Studia, nebude možné kvůli politice Microsoftu WinRT solusion spustit. Pokud bude potřeba zadat vývojářskou licenci, poskytnu ji dočasně přes email (richard.juchelka@gmail.com), jelikož je svázána s mým vývojářským účtem u Microsoftu.
- Pro otestování komunikace mezi dvěma online klienty je nutné spustit každého klienta na zvláštním počítači.

6 Závěr

Pro mě byl vývoj na WinRT platformě obrovskou zkušeností. Získal jsem základní dovednosti v oblasti tvorby uživatelského rozhraní pomocí XAML-u a určitě v budoucnu využiju i vývojový model MVVM, který opravdu zjednodušuje práci a zpřehledňuje kód. Nemalou zkušenost jsem získal i s principy WCF, praktického nasazení algoritmů RSA a SHA.

Než jsem s prací začal, byl jsem z modelu WinRT, respektive Metro stylu aplikací, nejistý v názoru, jestli je to opravdu použitelné a nebo v současné době vhodné. Microsoft postupem času stále sbližuje WinRT pro telefony a WinRT pro desktop a kdybych začal vyvíjet celou aplikaci znovu, již bych vytvořil společnou verzi pro oba typy API. V poslední verzi Visual Studia Microsoft sám programátorům toto předhazuje předvytvořenými projekty pro obě platformy v rámci jedné solusion. A takový typ aplikace by už měl opravdu dopad a smysl. Uživatel by mohl přecházet mezi zařízeními a snadno sdílet data mezi nimi.

Je tedy vidět, že tento model vývoje Microsoft myslí opravdu vážně a za sebe mohu říct, že díky postupnému pronikání do problematiky se mi principy WinRT líbí.

Richard Juchelka

7 Reference

[1] MICROSOFT. MSDN Blogs [online]. 2014 [cit. 2014-07-20]. Dostupné z: http://blogs.msdn.com/

[2] INTEL. Developer Zone [online]. 2014 [cit. 2014-07-20]. Dostupné z: https://software.intel.com

[3] MICROSOFT. Microsoft developer network [online]. 2014 [cit. 2014-07-20]. Dostupné z: http://msdn.microsoft.com

[4] Ján Hanák *Vývoj moderných WinRT-programov pre systém Windows 8*, 2012.